Find index of 1's in bitset and Encode/Decode positions using de-Bruijn Sequence

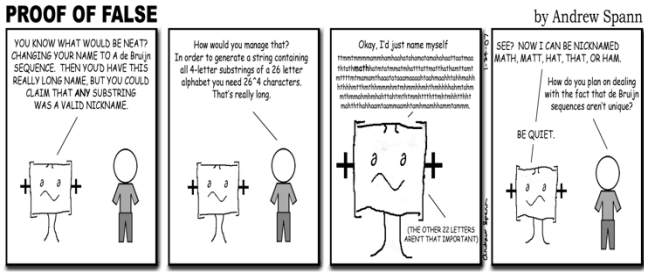
Elvan Owen and 13513082[1]
Program Sarjana Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*vanzz_95@hotmail.com*

***Abstract*— Brute force is a term we often heard of. It is a way of solving things by trying every possible solutions until it founds one. Mostly, we saw and heard of this term in internet or television or radio when people are talking about security where people are trying to break into system by brute forcing all the possible passwords. Nevertheless, there exist more efficient way to do brute force, which is using de Bruijn Sequence. Besides, there are many other useful application of de Bruijn Sequence .**

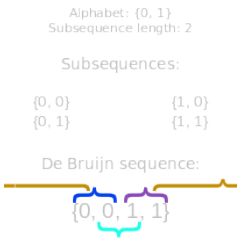
***Keywords*— Combinatorics, De Bruijn Sequence, Graph, Magic.**

I. Introduction

De bruijn Sequence is a sequence where every substring is different from one another .



We often have a situation where we have a system and need to test all possible state the machine can go. It would seem too tiring to try all the possible combinations one by one by typing each and every combinations. By using de Bruijn Sequence, we can try all possibility by effectively cutting off the number of presses needed. For example, to try all the possible combination of 2-bit string, we could try starting the lowest, in this case 00. Then we proceed to 01, 11, and finally 10. Therefore, de Bruijn Sequence is 0011, where we can take two characters out of the string continuously and generate all possible sequence.



de Bruijn Sequence is usually written as

$$B(k,n)$$

where k is the symbols in the alphabet and n is the length of the substring. Example given above can be stated as $B(2,2)$ where the 2 alphabets are $\{0,1\}$ and the length of the substring is 2.

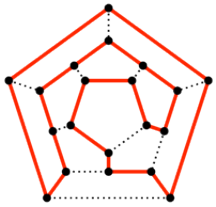
II. Basic Theory

One of the common things that happened in our life is to count how many different outcomes out of a set of things ie. To choose k items out of n items , we can state it as $C(n,k)$ or $\binom{n}{k}$ where

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Moreover, if we have n items and every items have k different state in which every items is independent of the others, then we have n^k different states.

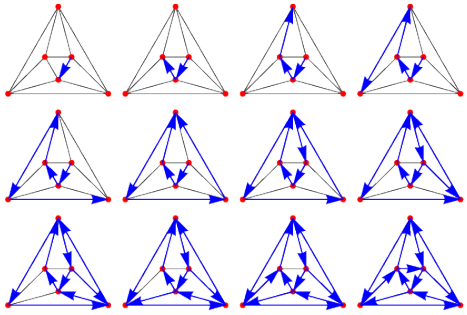
Hamiltonian Path is a path in a graph where every vertex in the graph is included in the path only once. Figure below shows path that includes every vertex in the graph.



Eulerian Cycle is a path in a graph where every edge in the graph is included in the path only once and the end of the path goes back to the start of the path. For directed graph, every vertex in the graph has to be connected and has equal in-degree and out-degree in order to have Eulerian Cycle property.

The alphabets below show the order of traversal of all

edges in the graph.



Here we are going to count how many possible strings out of n-bit strings and how long is the minimum de Bruijn sequence needed to represent all of the possible sequence . For example, if we want 3-bit substring with alphabet consisting only $\{0,1\}$, then de Bruijn sequence is

00010111 ……. (1)

There are 2 alphabets and each alphabet is independent of the others, therefore there are $2^3 = 8$ different 3-bit substrings :

1. 000
2. 001
3. 010
4. 101
5. 011
6. 111
7. 110
8. 100

However, we should realize that this de Bruijn sequence is not unique since we can have different sequences by reversing or rotating sequence (1) , ie. Reversing sequence (1) gives

11101000……. (2)

with substrings :

1. 111
2. 110
3. 101
4. 010
5. 100
6. 000
7. 001
8. 011

The minimum length of a de Bruijn sequence is k^n since there are k^n different substrings exist and all of them start at different points in the sequence. There are $\frac{k!^{k^{n-1}}}{k^n}$ different de Bruijn sequences. This number corresponds to the number of hamiltonian path exist in the graph.

III. CONSTRUCTING DE BRUIJN SEQUENCE

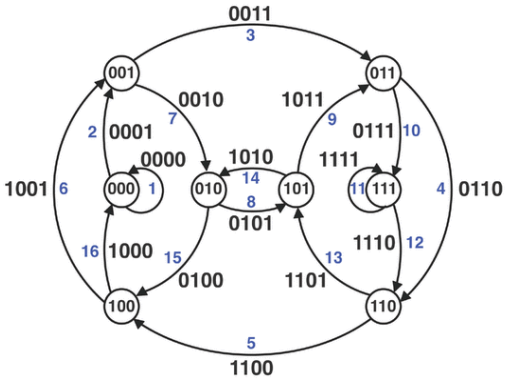
A. *Algorithm*

```
public class DeBruijn {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int N = 1 << n;                          // 2^n

        String deBruijn = "";
        for (int i = 0; i < n; i++)
            deBruijn = deBruijn + "0";

        for (int i = n; i < N; i++) {
            String suffix = deBruijn.substring(i - n + 1);
            if (deBruijn.indexOf(suffix + "1") == -1)
                deBruijn = deBruijn + "1";
            else
                deBruijn = deBruijn + "0";
        }
        System.out.println(deBruijn);

    }

}
```

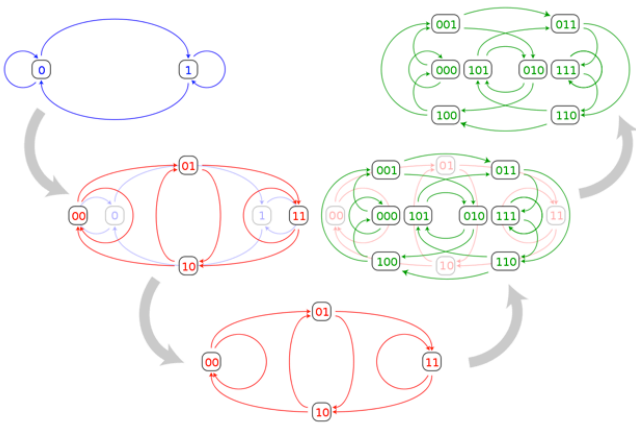

In the above algorithm, we start with n 0's and trying to append "1" to the end of the string if the substring has not existed, else we append "0".

B. *De Bruijn Graph*

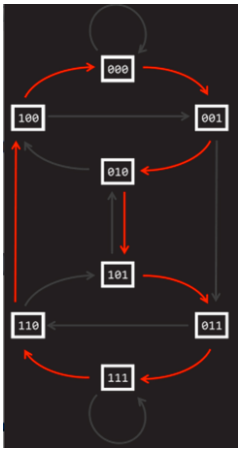
De Bruijn graph is basically a directed graph with edges as alphabets and vertices as length - n substring. There are k^n vertices and each of them have k out edges (alphabets) , example shown below.



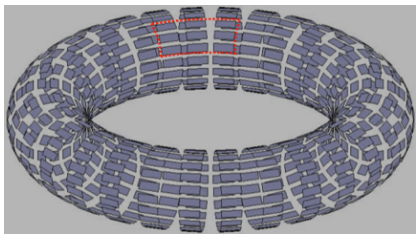
De Bruijn sequences can be created by traversing all vertices known as Hamilton path or by Eulerian cycle in $(n-1)$ de Bruijn Graph (Graph with substring length $(n-1)$) .Figure below shows relation between n and $(n-1)$ de Bruijn Graph .



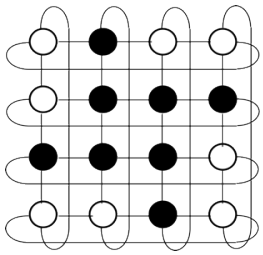
In the figure above, we can see that n De Bruijn Graph can be constructed by replacing every edges in $(n-1)$ de Bruijn Graph with new vertex and vice versa. Therefore, to create n de Bruijn Sequence, we can traverse all vertices in n de Bruijn Graph (Hamiltonian Path) or we can create $(n-1)$ de Bruijn Graph and traverse every edges (Eulerian Cycle) or. Figure below shows the Hamiltonian path in a length 3 de Bruijn Graph.



However, de Bruijn sequence is not limited in 1-dimensional. There is 2-dimensional de Bruijn sequence, known as de Bruijn Torus / Bruijn Arrays, can be imagined as sequence of matrices. It is called torus since every side of a matrix is connected to other matrices. Figure below gives more detail picture.

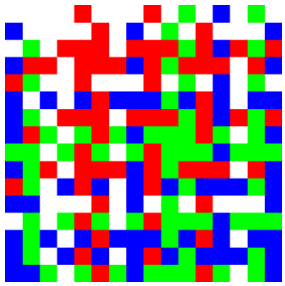


Or they can be viewed in 2-dimensional as



The figure above represents $B(4,4\,;2,2)$,where the size of the matrix is $4\,x\,4$ and the de Bruijn matrix (window) is $2\,x\,2$. Take any $2\,x\,2$ subarray (window) in the matrix above and they all represent different possible subarrays.

Below are example of $B(16,16\,;2,2)$, where there are 256 different $2\,x\,2$ window .



IV. APPLICATIONS OF DEBRUIJN SEQUENCE

1. Finding index of a 1 in a word

We know there is a lot of ways to do this such as traversing and comparing one by one the bits in a word starting from the Least Significant Bit. Since until today the biggest number of word size is still small, using this de Bruijn sequence does not matter much, but imagine when you have a word size 1000 or more. Traversing one by one will start becoming slow.

The first step to solve this problem using de Bruijn sequence is to create consecutive indexes $0\,,1,..,n-1$. (n : word size) and then hash it with length-n de Bruijn Sequence and then map each substring length $ceil\,(\,log2(n)\,)$ with each indexes $0\,,1,..,n-1$. Example shown below with $B(k,m)$: $B(2,3)$ with alphabets $k:\{0,1\}$ and length of sequence $m{:}\,log2(n)$, where n is 8-bit word. $B(2,3)$:

`debruijn = 00011101`

$h(\mathbf{x})$	Index
000	0
001	1
010	6
011	2
100	7
101	5
110	4
111	3

Consider D: deBruijn sequence, W: original word, B: modified word with only LSB left, LSB: Least Significant Bit.

The second step is to isolate the last set bit in the word. We can use bit techniques

$$B = W\&{\sim}W$$

to isolate W's LSB. Then simply mutiply B with D. What actually happens when we multiply B with D ? We are shifting y bits in D, where y is the position of B's only 1 or W's LSB.

Afterward, the last step is to take starting from MSB $log2(n)$ bits and check it with the hash tables to obtain the index of the W's LSB .Recur these steps until W equals 0.

Below is the algorithm for all the steps above:

```
#define debruijn32 0x077CB531UL
/* debruijn32 = 0000 0111 0111 1100 1011 0101 0011 0001 */

/* table to convert debruijn index to standard index */
int index32[32];

/* routine to initialize index32 */
void setup( void )
{
  int i;
  for(i=0; i<32; i++)
    index32[ (debruijn32 << i) >> 27 ] = i;
}

/* compute index of rightmost 1 */
int rightmost_index( unsigned long b )
{
  b &= -b;
  b *= debruijn32;
  b >>= 27;
  return index32[b];
}
```

Figure 1: 32-bit C implementation of the DEBRUIJN strategy.

2. Decode and Encode positions using de Bruijn Sequence

I believe that everyone has ever seen magician showing card tricks. There is one card trick I have ever seen and it was kinda cool. You are given a deck of cards, then you can cut the deck as many times as you want. Then the magician can guess the cards relative to your card positions or some other tricks with the same idea. Basically, the card sequence can be seen as de Bruijn Sequence with a little symbols encoding.

The first step is to create de Bruijn sequence based on your deck of cards. In this example we'll consider only 32 cards with $A, 2, 3, 4, 5, 6, 7, 8$ and each have 4 suits: ♣ ♠ ♦ ♥ . Since there are only 8 different cards, 3 bits will be enough to cover all different cards + 2 bits to cover 4 different suits types. Therefore we have a total of 5 bits and we can create length-5 de Bruijn sequence :

$$00001010111011000111110011010010$$

We can then encode the card in this way : suits + numbers ie. 2♣ can be encoded as : ♣ → 00 + 2 → 001 = 00001.

00	♣	000	A	100	5
01	♠	001	2	101	6
10	♢	010	3	110	7
11	♡	011	4	111	8

Therefore, the sequence of the deck by following above sequence becomes :

♣2, ♣3, ♣6, ♠3, ♢6, ♠4, ♢8, ♠7, ♡6, ♡4, ♢7, ♠5, ♡A, ♢2, ♣4, ♣8, ♠8, ♡8, ♡7, ♡5, ♡2, ♢4, ♣7, ♠6, ♡3, ♢5, ♠2, ♢3, ♣5, ♠A, ♢A, ♣A

Here we can see that regardless how many times the deck is being cut, the magician can merely memorize the 32 long bits sequence and identify all the cards prior to or after a relative card. Here the magician can simply ask the player to look at the card from the bottom of the deck and do the rest of the magic depending on how the magician wants to finish it since he knows all about the rest of the deck.

Furthermore, there is a technology known as digital paper, along with digital pen. How it works is closely related to de Bruijn arrays, where the paper contain de Bruijn pattern and whenever someone writes something into the paper, the digital pen scans the pattern and sends signals about the pattern to the computer and the computer will try identify the positions and knows what is written and in the end, prints it to the screen .

VI. ACKNOWLEDGMENT

I want to thank Mr. Rinaldi and Mrs. Harlili for being such a great Discrete Math teacher for us, who have taught us a lot of things for a semester. Without them, I may not have written this paper. Thank you teachers…

REFERENCES

[1] http://supertech.csail.mit.edu/papers/debruijn.pdf
[2] http://web.mnstate.edu/goytadam/talks/DBS.pdf

[3] http://www.math.toronto.edu/ddmoskov/mat332/Bruijn.pptx
[4] https://www.math.ubc.ca/~anstee/math443/DeBruijnCardTrick.pdf
[5] http://www.datagenetics.com/blog/october22013/index.html
[6] http://introcs.cs.princeton.edu/java/31datatype/DeBruijn.java.html
[7] http://www.nature.com/nbt/journal/v29/n11/images_article/nbt.2023-F2.gif
[8] http://en.wikipedia.org/wiki/De_Bruijn_sequence
[9] http://en.wikipedia.org/wiki/De_Bruijn_graph
[10] http://en.wikipedia.org/wiki/De_Bruijn_torus
[11] http://en.wikipedia.org/wiki/Digital_paper
diakses pada tanggal 8 December pukul : 20.00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 December 2014

Elvan Owen 13513082